1

A SYNCHONIZATION METHOD IN AN N-TIER TRANSACTION

Field of the Invention

The present invention relates to the field of synchronization and more particularly, to synchronization in an n-tier architecture.

Background of the Invention

The Internet is changing many aspects of our lives, but no area is undergoing as rapid and significant a change as the way businesses operate. For businesses today, Internet technology is no longer an afterthought in forming a business strategy but has become a driving force. e-business is defined by many organizations as the transformation of key business processes through the use of Internet technologies. An e-business connects critical business systems directly to customers, employees, suppliers, and distributors via the Internet to improve time to market, access a broader base of customers and suppliers, improve efficiency, and reduce costs.

Examples of companies implementing an e-business model include retailers offering online purchasing integrated with online supply chain management or electronic financial service organizations that reduce the high cost of transactions while providing improved access to customer accounts. e- business has progressed from being a means of automating certain business functions to become an essential element of competitive business.

A framework is a reusable design expressed as a set of abstract patterns and the way their interfaces collaborate. It is a reusable design for all or part of a software system; a user interface framework only provides a design for the user interface of a system while an application framework provides a design for the entire application.

Application frameworks are becoming increasingly important for developing complex applications. An application framework describes a set of interacting components and services available to an application, the responsibilities of and the interactions between the components and services. A developer creates an application by composing and extending the components and services available in the application framework.

Application frameworks typically address specific business domains, such as manufacturing or finance.

Early frameworks revolved around programming languages, such as an object-oriented (OO) design framework. Application frameworks do not have to be implemented in an object-oriented language, but they do have characteristics similar to OO design in that each component of the framework has defined interfaces and behaviour. An application framework provides a context for the software, servers, and services necessary to create, deploy, and manage complex e-business applications.

The application framework used by International Business Machines Corporation for e-business provides a model for designing e-business solutions. This model has evolved from the traditional client/server computing model and is based on an "n-tier" distributed environment where tiers of application logic and business services are constructed from components that communicate with each other across·a network.

In its most basic form, the framework can be depicted as a logical three-tier computing model meaning that there is a logical, but not necessarily physical, separation of processes. With reference to FIGURE 1, there is shown an example of a three-tier model (100) comprising a client tier (105), a middle tier (110) and a third tier (115).

The client tier (105) comprises clients (e.g. smartcards, digital wireless telephones, personal digital assistants (PDAs)) having logic related to sending requests to applications (e.g. through a browser or Java applet) and to presenting information and results produced by an application to the user (via a graphical user interface). The clients are sometimes referred to as "thin" clients, meaning that little or no application logic is executed on the client and therefore relatively little software is required to be installed on the client. Typically, only user interaction and input validation functions run on the client. Clients are implemented with industry-standard technologies and protocols (e.g. TCP/IP, HTTP, HTML/DHTML/XML, and Java (Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both)) that enable them to interact with the user, communicate with a middle tier and send and receive standard data formats.

The use of thin clients improves manageability, flexibility, and time to market. Advantageously, a broader range of client devices can be

supported since the dependency on device capabilities are reduced.
Furthermore, user's application environments can be centrally managed and
distributed to client devices of different types and in different
locations to provide support for mobile users.

5

The middle tier (110) comprises application servers, which are
single application or multi-application servers. In an e-business
environment, the servers are web application servers that are optimized
for Internet applications. The application servers are the platforms that
10   provide the run-time environment for an application's business logic. The
business logic is executed independently of the client type and user
interface style. The separation of presentation logic from the application
logic enables the creation of reusable application components that can be
used from a number of different styles of user interface.

15

The middle tier servers incorporate several application integration
technologies for communicating with applications, data, and services in
other tiers. Web application transaction servers are essential in the
e-business infrastructure because the number of online transactions is
20   increasing tremendously. Customers need high-speed transaction processing,
robust system reliability, backup and quick recovery of mission-critical
applications. Web application servers must also provide a comprehensive
integration capability with other systems. They must also connect with
each other as well as the back-end third tier to enable these new business
25   processes.

The web application servers are implemented using various Internet
and Java technologies, including the HTTP server and the platform
independent Enterprise Java programming environment. The latter eliminates
30   the dependence of the business logic on the underlying hardware, operating
system and networking infrastructure, thus increasing portability and
decreasing development and maintenance costs. The middle tier incorporates
the network infrastructure and foundation services that enable rapid
development and deployment of applications in a distributed network
35   environment. Applications (120) run as a combination of servlets,
server-side scripts, and Enterprise JavaBeans (EJBs) in the web
application server and its Java Virtual Machine. EJBs provide much of the
business logic of an application, particularly access to database and
transaction services. EJBs isolate the developer from the unique
40   characteristics of the underlying database and transaction services,
simplifying the development of platform independent business logic.

4

The third tier (115) comprises legacy systems that have been used
for many years (hence the term "legacy") that support new and existing
internal applications, services and data; and also external applications,
services and data from new and existing business partners. Connections to
these services leverage value for customers, business partners and
employees. External services and legacy systems are fundamental to the
emergence of the web application model because they are the result of
years of corporate investment in information technology. These are the
mission-critical applications and data that everyone depends on daily, and
they are the business assets that need to be made available to the Web in
a secure, controlled way to enable companies to leverage their value for
customers, employees, and suppliers in intranets and extranets. The
servers in this tier provide the data storage and transactional
applications used by the web application server processes.

Application elements residing in these three logical tiers are
connected through a set of industry-standard protocols, services, and
software connectors. The connectors (125, 126, 127) connect the new,
value-add business logic in the middle tier to the vast accumulated assets
in a company's existing applications and data systems in the back-end
tier. The business logic running on the middle tier accepts a request
(e.g. HTTP request) from the client and invokes a connector to securely
communicate with the back-end tier services on the client's behalf.

An alternative model is the "data synchronization" model. With
reference to the distributed system (200) of FIGURE 2, an application
resides on a server (215) (e.g. in storage (220)) and a replica of the
same application resides on a "fat client" (e.g. a personal computer)
(205) (e.g. in storage (210)). The fat client executes most or all of the
processing of applications itself. When a user is offline, they can work
locally on the replica data and at intervals, the user connects to the
server (215), via a network (225) and the replica data is synchronized
with the master data residing on the server (215). This model is often
used for Personal Information Management (PIM) applications e.g. calendar,
email etc and is particularly useful for mobile clients, which are only
intermittently connected to the network.

Currently, many organizations would like to include data
synchronization as a method to deliver e-business applications. For
example, banks would like to provide customers with the ability to view
and manipulate their accounts from a PDA application, with regular
synchronization of the local data on the PDA with the master data on the

back-end tier. However, the data synchronization model does not fit well
with the e-business three-tier model. Specifically, organizations are
unwilling or unable to synchronize their complex, highly protected
back-end databases directly with the clients. Another complexity is that
the business logic of applications running on a web application server, is
driven by HTTP requests and produces HTTP responses. Synchronization
protocols that are required between a client and the back-end tier are not
understood by the business logic.

One possible solution is to extract a copy of the data from the
back-end tier to the middle tier. The clients can then connect to the
middle tier and update the data. The updated data can then be synchronized
with the data stored on the back-end tier via the middle tier.

However, this solution has disadvantages. Firstly, the data that the
client works on will always be out of date, especially if the data is
extracted in an overnight batch. Furthermore, new application function
must be developed to synchronize the updated data with the data on the
back-end tier in order to maintain coherent data on the back-end tier.
This in turn results in the need for distributed transaction processing,
which has well-known drawbacks. For example, if a transaction coordinator
"commits" a transaction (e.g. Bank A debits money from Account A and Bank
B credits the money to Account B), to all resource managers that are
participating in that transaction but then the network were to go down,
the coherence of the state of the transaction can not be relied upon. Some
of the resource managers would register the transaction as being
completed, but some may not. Typically, the resource managers that require
clarification on the outcome of the transaction need to contact the
transaction coordinator. However, this is not possible until the network
is live again and in the meantime, Accounts A and B could be locked for
some time. Another disadvantage is that to scale the solution to several
thin clients, the middle tier must typically comprise several web
application servers. The problem of maintaining coherency therefore
dramatically increases.

US Patent No. 6.023.684 discloses a three tier model, whereby at the
initiation of a logical session with a client program, an application
service refreshes data for a customer associated with the client program,
by using data obtained from a back-end processing system through the host
interface. The data is stored in local data memory associated with the
application service and this data is then used by the application service
for processing client requests during the logical session. The local data

6

memory permits the processing of the client request to be decoupled from the updating of the back-end processing system.

Thus there is a need for a model that provides the benefits of the data synchronization model and the "n-tier" model, without the need for extensive change of the "n-tier" model.

Disclosure of the Invention

According to a first aspect, there is provided a method of synchronization for use in a distributed data processing system comprising: at least one legacy computer having means for storing a master version of data, a first non-legacy computer having means for supporting synchronization, and a second non-legacy computer having means for storing a copy of said master version of data and means for executing at least one operation on said copy, said method comprising the steps of: executing, by said second non-legacy computer, said at least one operation on said copy, sending, by said second non-legacy computer, said at least one operation to said first non-legacy computer, executing, by said first non-legacy computer, said at least one operation on said master version at said at least one legacy computer, determining if said executing step is successful, and in response to a successful executing step, synchronizing said master version by applying said at least one operation.

As an example, the legacy computer is a mainframe computer, the fist non-legacy computer is a web application server and the second non-legacy computer is a PDA. Preferably, the method further comprises the step of: sending, by the second non-legacy computer, a synchronization protocol to the first non-legacy computer. The protocol will typically be particular to the type of second non-legacy computer. In a preferred embodiment, if there are two or more operations to be executed on the legacy computer, the operations are executed in sequence.

Aptly, once the at least one operation has been executed on the legacy computer, the first non-legacy computer sends the results (i.e. success/failure notification as well as the resulting data itself) and a new copy of the master version of data to the second non-legacy computer. Preferably, if the at least one operation cannot be executed on the master version (e.g. because of conflicting concurrent operations), master version is not synchronized.

According to a second aspect, there is provided a distributed data
processing system for synchronization comprising: at least one legacy
computer having means for storing a master version of data, a first
non-legacy computer having means for supporting synchronization, and a
second non-legacy computer having means for storing a copy of said master
version of data and means for executing at least one operation on said
copy, said system further comprising: means for executing, by said second
non-legacy computer, said at least one operation on said copy, means for
sending, by said second non-legacy computer, said at least one operation
to said first non-legacy computer, means for executing, by said first
non-legacy computer, said at least one operation on said master version at
said at least one legacy computer, means for determining if said executing
step is successful, and means, responsive to successful determination, for
synchronizing said master version by applying said at least one operation.

According to a third aspect, there is provided a computer program
comprising computer program code means adapted to perform all the steps of
the above method when said program is run on a computer.

Brief Description of the Drawings

The present invention will now be described, by way of example only,
with reference to preferred embodiments thereof, as illustrated in the
following drawings:

FIGURE 1 is a schematic representation of a prior art "n-tier" model;

FIGURE 2 is a schematic representation of a prior art data synchronization
model;

FIGURE 3 is a schematic representation of an "n-tier" model, in accordance
with the present invention;

FIGURE 4 is a flow chart showing the operational steps involved in a data
synchronization process, implemented in the model as shown in FIGURE 3;

FIGURE 5 is a sequence diagram of the flows involved in the data
synchronization process between the components in the model as shown in
FIGURE 3; and

FIGURE 6 is a representation of the results of data synchronization.

Detailed Description of the Invention

     FIGURE 3 shows a pictorial representation of a three-tier model
(300) in which the present invention may be implemented. There is shown a
client tier (305), a middle tier (315) and a back-end tier (325) having
associated storage means (310, 320 and 330 respectively). The client tier
(305) and middle tier (315) are connected by a software connector (331)
and the middle tier (315) and back-end tier (325) are connected by
software connectors (332 and 333). The applications and services on the
back-end tier (325) are protected by a firewall (335).

     One embodiment of the present invention will now be described with
reference to FIGURES 3, 4 and 5. An example of a banking application is
described, whereby the thin client (305) is a PDA, the middle tier (315)
is a web application server and the back-end tier (325) is a data
repository for storing master versions of bank account data.

     Preferably, the thin client (305) comprises a function, which when
executed, obtains a copy of a user's bank account data from the storage
means (330) of the back-end tier (325). Preferably, the copy is stored
locally in the storage means (310) associated with the thin client (305).
In this example, bank account data with a balance of £300 is stored
locally. The user executes (step 400, 500) operations (e.g.
changes/updates) against the local data. In this embodiment, the
operations are:

- A monetary transfer of £50 into the account

- Setting up a standing order of £100 out of the account

- Making a one-off payment of £75 out of the account

     However, the local data is more than likely to be outdated with
respect to the master version residing on the back-end tier (325). This is
because some of the user-initiated operations may clash with concurrent
operations being applied to the master version of the user's bank account
at the back-end tier (325). For example, standing orders being paid,
monetary transfers to savings accounts, mortgage payments, etc. In this
embodiment, a concurrent operation has been applied to the master version
of the user's bank account. In this example a one-off payment of £200 has
been taken out of the user's bank account and therefore, the actual
balance stands at £100 (i.e. £300 – £200 = £100).

Since the operations initiated by the user have not yet been reconciled with any concurrent operations that have been applied to the master version, preferably, the user's operations are retained in a list. The thin client (305) therefore stores two types of data, namely, a copy of the "current" (possibly slightly divergent from the master version) data (i.e. the user's bank account data) and a list of pending operations (i.e. a list of operations 1, 2 and 3 above).

Next, the user at the thin client (305) sends (step 405, 505) a request (Request A) to synchronize the updated local data with the master version via an HTTP request to the middle tier (315). The middle tier (315) acknowledges receipt of "Request A" by sending (step 410, 510) back an HTTP response (Response A') instructing the thin client (305) to proceed with synchronization. It should be understood that this request/response exchange is an optional stage. Next, the user at the thin client (305) sends (step 415, 515) an HTTP request (Request B) to the middle tier (315) in order to start synchronization. The thin client's standard synchronization protocol is also sent to the middle tier (315). Advantageously, in addition to exploiting the business application logic on the middle tier by using the HTTP request/response model, the functions of the middle tier, which can run native thin-client synchronization protocol(s), are also exploited.

The synchronization process now starts (step 420). Specifically, the data to be synchronized is the list of operations (1, 2 and 3 above) that the user at the thin client (305) would like to perform. In the synchronization step, the list of operations is loaded from the thin client (305) to the middle tier (315). The middle tier (315) then replays (520) the operations in the list on the back-end tier (325), that is, a monetary transfer of £50 is made into the account, a standing order of £100 out of the account is set up and a one-off payment of £75 is made out of the account. It should be understood that preferably, the replaying step occurs promptly, so that the chances of conflicts are decreased. Also, preferably, the list of operations is replayed sequentially.

It should be understood that some of the user-initiated operations may succeed and some of them may fail, depending on whether any conflicting concurrent operations have been applied to the master version residing on the back-end tier (325). If concurrent operations have been applied to the master version, these would have been applied *after* the thin client (305) had received a "current" copy of the data. At step 425,

for each operation, a determination is made as to whether it is successful
or not. In response to a positive result, the master version of the data
is synchronized (step 430) to reflect the successful user-initiated
operations and in response to a negative result, the master version of the
data is not synchronized (step 435). In this example, the actual bank
account balance stands at £100. Therefore, operation 1 succeeds (after
which, the actual bank account balance = £150), operation 2 succeeds
(after which, the actual bank account balance = £50) and operation 3 fails
(because the actual bank account balance would = £-25).

Next, the back-end tier (325) sends (525) the results of the
user-initiated operations to the middle tier (315) as well as a copy of
the new (updated) master version of the bank account (in this example, the
actual bank account balance = £50).

The middle tier (315) monitors (step 440) the results of these
operations and sends (step 445, 530) the results and also the copy of the
new master version of the bank account to the thin client (305). The new
master version of the bank account replaces the local copy that was used
by the user in step 400.

The synchronization process has now ended and this is indicated to
the user by sending (step 450, 535) an HTTP response (Response B') from
the middle tier (315) to the thin client (305). Preferably, the user can
then be presented with an updated view of the state of the bank account, a
list of the successful operations and a list of failed operations. An
example of the view (600) is shown in FIGURE 6. At step 455, if no further
operations are to be carried out, the process ends. However, in response
to a positive result, processing returns to step 400 where, for example, a
user can act upon the failed operations by canceling them, modifying them,
deferring them etc.

Advantageously the middle tier only needs to access the master
version of the data after synchronization has occurred, in order to send
the thin client an up-to-date copy of the new master version of the data.
Furthermore, the preferred embodiment accommodates the possibility that
concurrent operations might have been applied to the master version of the
data. This scenario is handled without the need for consistent copies of
the data at the thin client and the back-end tier.

Another advantage is that the benefits of a data synchronization
model are provided without the need to extensively change the three-tier
model. Therefore, the present invention can be utilized in systems that

are already supported by many organizations today. For example, the
existing relationship between the thin client and middle tier can be used,
namely, the currently available HTTP and PDA-synchronization protocols.
However, it should be understood that the present invention could be
·5   implemented with other protocols. Furthermore, although a three-tier model
has been described, the present invention could be implemented in a model
with "n" tiers. Also, although a PDA, web application server and back-end
host have been described, it should be understood that the tiers could
comprise any other computer machine.
10